Willi Sicking

NDR-Klein-Computer erfaßt Meßwerte

Der NDR-Klein-Computer ist in der Ausbaustufe mit der CPU 68008 und Pascal-Compiler zwar in erster Linie für Aus- und Weiterbildungszwecke gedacht, jedoch läßt er sich auch hervorragend zur Meßwerterfassung und -auswertung im Labor einsetzen. Dabei kommen dem Anwender der klare Systemaufbau, die unbenutzten Interrupteingänge, die Grafik und die Dokumentation zugute. Um jedoch einen Rechner effektiv und komfortabel in der Meßwerterfassung einsetzen zu können, ist eine gute Verbindung zwischen Assemblerebene und Hochsprache nötig.

E wie Execute

Vom Pascal-Programm aus sind drei Möglichkeiten vorgesehen, mit der Assemblerebene Kontakt aufzunehmen. Beispiele zeigt *Bild 1*. Mit dem Steuerzeichen CHR(1) wird dem Compiler mitgeteilt, daß es sich um eine Ausgabe an das System handelt. Folgt darauf E 'Name', wird das Unterprogramm „Name" aufgerufen, und die nachfolgenden Variablen oder Konstanten werden an die Register D0 bis D7 übergeben. Anstelle des Namens kann man auch direkt eine Adresse oder durch Voransetzen des @-Zeichens eins der 65 Systemunterprogramme ansprechen. Übergeben kann man nur Integergrößen.
Leider ist die Übertragung in umgekehrter Richtung nicht auf gleiche Weise möglich. Dazu muß man sich des G wie Get bedienen. Der Inhalt der Adresse wird beim nächsten Read an die entsprechende Variable übergeben. Ein dem „Poke" vergleichbarer Befehl wir mit P 'Adresse', Variable aufgerufen. Anstelle direkter Adressen sind auch hier symbolische Namen möglich, die vorher vom Assembler vereinbart wurden. Die Funktion Get dauert 5 ms, Poke nimmt 4 ms in Anspruch. Es wird nur jeweils ein Byte übergeben. Die Ausgabe des Programms in Bild 1 wäre also 1.

Gleitkomma- und Festkommaformate

Intern berechnet der NDR-Klein-Computer Gleitkommazahlen in einem 8 Byte langen BCD-Format. Diese Darstellung erlaubt zwar keine superschnelle Arithmetik, sie ist jedoch sehr interfacefreundlich. Berechnete Größen kann man leicht an ein 7-Segment-Display anpassen, und BCD-Meßwerte können einfach in das Gleitkommaformat transformiert werden, wie später gezeigt wird. In *Bild 2* sind die Variablen A=10 und

```
009500                  ORG $9500
= 00035290              FADD EQU $35290
= 000352B4              FSUB EQU $352B4
= 000352FE              FMUL EQU $352FE
= 0003536C              FDIV EQU $3536C
009500
009500
009500  00000000        OUT: DC.L 0,0
009504  00000000
009508  01000000        A:   DC.L $01000000,$00000082 ; =10
00950C  00000082
009510  03000000        B:   DC.L $03000000,$00000081 ; = 3
009514  00000081
009518
009518                  START:
009518  41F9 00009500   LEA.L OUT,A0
00951E  45F9 00009508   LEA.L A,A2
009524  43F9 00009510   LEA.L B,A1
00952A
00952A  303C 07D1       MOVE.W #2001,D0 ; 2001 MAL
00952E
00952E                  LOOP:
00952E  2092            MOVE.L (A2),(A0)
009530  216A 0004 0004  MOVE.L 4(A2),4(A0)
009536  4EB9 0003536C   JSR FDIV ;RECHNET 10/3
00953C  51C8 FFF0       DBRA D0,LOOP
009540  4E75            RTS
009542

0000    Fehler entdeckt
008ABE  Ende-Symboltabelle
```

Bild 2. mc-Benchmarktest in der Assemblersprache des 68000

```
**********************************
* PASCAL/S  Pcode-Compiler V3.1 *
* (C) 1984 Rolf-Dieter Klein    *
* Version nach PASCAL/S von     *
* N.Wirth 1976, E.T.H Zuerich   *
**********************************

    0  program demo(input,output);
    0  var i:integer;
    0
    0
    0
    0  begin
    0  writeln(chr(1),'E $drawto ',0,' ',300,' ',200);
```

```
16  i:=1025;
19  writeln(chr(1),'P $9500 ',i);
27
27
27
27  writeln(chr(1),'G $9500 ');
33  read(i);
35  writeln(i);
38
38
38  end.
```

Bild 1. Die drei Möglichkeiten, Daten zwischen Pascal- und Assembler-Programm auszutauschen

B=3 direkt im Gleitkommaformat eingegeben. Negative Zahlen werden mit einer 9 in der ersten Stelle gekennzeichnet. Die BCD-Stellen enthalten dann das Zehnerkomplement. Die letzten beiden Digits bilden den Exponenten. Zum Beispiel wird dann –3456 zu 9654400000000084. Natürlich läßt sich mit den vorhandenen Gleitkommaroutinen auch hervorragend in Assemblersprache rechnen. Ein Beispiel für einen Benchmarktest zeigt *Bild 2*. Im Prinzip handelt es sich um den mc-Test „2001 Divisionen 10/3". Beim Aufruf einer Gleitkommaroutine müssen A0 und A1 auf den Input zeigen, das Ergebnis wird wieder unter A0 abgelegt. Daher schiebt das Programm dauernd (A2) nach (A0). Das Ergebnis ist 13stellig und liegt nach neun Sekunden vor, wenn das Programm im DRAM läuft. Im SRAM geht es wegen der fehlenden Refresh-Wartezeit knapp 20 % schneller.
Bild 3 zeigt den gleichen Test in Pascal. Hier dauert es im DRAM 13 Sekunden, im SRAM elf Sekunden. Wie man sieht, arbeitet der P-Code-Interpreter ganz ordentlich.
Das 16-Bit-Integerformat erlaubt Zahlen von +32767 bis –32767.
+32767 entspricht intern $7FFF,
–32767 entspricht intern $8001,
–1 entspricht intern $FFFF.
Die Speicherverwaltung des Pascal-Compilers reserviert jedoch für die Integer-Zahlen ebenfalls 8 Byte. Signifikant ist das erste Wort.

Erfassen in Assembler – weiterverarbeiten in Pascal

Bei der Meßwerterfassung und -auswertung dient das Assemblerprogramm in der Regel zur schnellen Aufnahme; mit dem Pascal-Programm werden die Daten bearbeitet, ausgewertet und dargestellt. Die 4 bzw. 5 ms Zugriffszeit sind jedoch für viele Anwendungen zuviel. Der

```
 0  program eva(input,output);
 0
 0  var a:real;
 0      i:integer;
 0
 0
 0  begin
 0  for i:=0 to 2000 do begin
 4  a:=10/3
 7  end;
12
12  writeln(a);
15  end.
```

Bild 3. Benchmarktest von Bild 2 in Pascal formuliert

```
 0  program test(input,output);
 0  var a,b,c:real;
 0      esc:char;
 0      i:integer;
 0  begin
 0  esc:=chr(1);
 4  for i:=1 to 1000 do begin
 8  writeln(esc,'E #9500 ');
13  end;
14  writeln(a);
17  writeln(b);
20  writeln(c);
23  writeln ;
24  writeln('die Frequenz betrug ',a:8:4,' KHz');
33  end.
```

```
Start PCODE-Interpreter 3.1
     4.825285900000e+03
     1.000000000000e-129
     0.???6;????57???>e+94

die Frequenz betrug 4825.2859 KHz
```

Bild 4. Der Variablen A wurde ein Meßwert untergeschoben

NDR-Klein-Computer hat nun bei der Verwaltung der Variablen eine seltene Eigenschaft: Deklarierte Variablen bekommen im Stack einen festen Speicherplatz zugewiesen, der jedoch beim Programmstart nicht automatisch gelöscht wird, so daß man dort bei einem Zugriff zufällige Ziffern- und Zeichenfolgen oder Werte aus früheren Programmen findet.
Dieses Verhalten kann man sich zunutze machen, indem man die Stackadressen der benötigten Variablen oder auch ganzer Felder bestimmt und mit dem Assemblerprogramm die Daten genau in diese Speicherstellen im richtigen Format ablegt. Auch die Speicherung auf Kassette und die Übergabe der Werte zwischen verschiedenen Programmen ist so möglich. Es ist nur darauf zu achten, daß die Anzahl und Reihenfolge bei der Deklaration übereinstimmen.

Auch ist so ohne zusätzliche Hardware eine interruptgetriebene Meßwerterfassung (Uhr, Ereignis) realisierbar, wobei eine Interruptroutine ein (Pascal-)Feld füllt. Dazu muß lediglich, je nach Priorität, einer der drei Autointerruptvektoren zur Aufnahmeroutine hin verbogen wer-

```
009500                  ORG $9500
= FFFFFF30              NIB EQU $FFFFFF30
= FFFFFF31              DIGIT EQU $FFFFFF31
= 0001C5AA              OUT EQU $1C5AA ;VAR A
009500  4280           CLR.L D0
009502  4281           CLR.L D1
009504  4283           CLR.L D3
009506
009506                  TOP:
009506  41F9 FFFFFF30  LEA.L NIB,A0
00950C  43F9 FFFFFF31  LEA.L DIGIT,A1
009512  243C 00000080  MOVE.L #$80,D2 ;DIGITVERGLEICH
009518                 WAIT:          ;WARTEN AUF DIGIT
009518  1011           MOVE.B (A1),D0
00951A  B002           CMP.B D2,D0    ;RICHTIGES DIGIT ?
00951C  6600 FFFA      BNE WAIT
009520  1210           MOVE.B (A0),D1 ;BCD ZIFFER LADEN
009522  E99B           ROL.L #4,D3
009524  8601           OR.B D1,D3     ;MIT ANDEREN VERBINDEN
009526  E29A           ROR.L #1,D2    ;NEXT DIGIT
009528  0C02 0000      CMP.B #0,D2
00952C  6600 FFEA      BNE WAIT       ;BIS ALLE DIGITS FERTIG
009530
009530  2803           MOVE.L D3,D4          ;
009532  0283 FFFFFFF0   ANDI.L #$FFFFFFF0,D3 ;
009538  E89B           ROR.L #4,D3           ; GLEITKOMMAFORM
00953A  0284 0000000F  ANDI.L #$F,D4         ;
009540  E89C           ROR.L #4,D4           ;
009542  0604 0084      ADDI.B #$84,D4 ; EXPONENT DAZU
009546  41F9 0001C5AA  LEA.L OUT,A0   ;
00954C  2083           MOVE.L D3,(A0) ; UNTERSCHIEBEN
00954E  2144 0004      MOVE.L D4,4(A0);
009552  4E75           RTS
009554

0000    Fehler entdeckt
008A88  Ende-Symboltabelle
```

Bild 5. Assemblerprogramm zur Frequenzmessung mit dem ICM7226A

den, die dann mit RTE anstatt mit RTS endet.
In *Bild 4* werden die drei Variablen A, B, C definiert und ausgegeben. A bekommt vom Assemblerprogramm (*Bild 5*) einen Gleitkommawert untergeschoben. B und C sind zur Demonstration nicht definiert. Mit E '$9500' wird das Assemblerprogramm aufgerufen. Das Ergebnis steht dann direkt in der Variablen A.

Frequenzmessung mit etwas Zusatzhardware

Der ICM7226A ist ein Einchip-10-MHz-Universalfrequenzzähler. Je nach Beschaltung von Anschluß 4 dient er zur Frequenzmessung von 0...10 MHz, zur Periodendauermessung von 0,1 µs bis 10 s, zur Frequenz- und Zeitverhältnismessung oder als Ereigniszähler. Normalerweise werden die acht Dekaden im Multiplexverfahren an eine 7-Segment-Anzeigeeinheit geschickt. Über zwei Schalter oder Drahtbrücken werden Betriebsart und Meßzeit eingestellt. Da der Baustein zusätzlich über einen gemultiplexten BCD-Ausgang verfügt, kann man den Meßwert leicht in den Rechner einlesen. Die Multiplexfrequenz beträgt 500 Hz, so daß innerhalb von 2 ms das Ergebnis übertragen ist.
Das Assemblerprogramm (Bild 5) wartet zunächst auf das MUX-Signal an D8 und übernimmt bei Erscheinen den BCD-Wert der ersten Stelle. Das Digit wird mit Register D3 ODER-verknüpft, das vorher um eine Stelle (4 Bit) nach links verschoben wurde. Nun wird der Digitzähler D2 von $80 auf $40 gesetzt und der nächste BCD-Wert geladen. Nach acht Durchläufen steht das Ergebnis in D3.
Der Rest des Programms erzeugt das Gleitkommaformat und fügt den passenden Exponenten dazu. Am Schluß wird die Gleitkommazahl auf den ersten Stackspeicher ($1C5AA) geschoben, der im Pascal-Programm (Bild 4) von der Variablen A belegt ist. Die gesamte Übertragung dauert 6 ms.
Die Frequenzmessung mit dem Rechner bietet viele Möglichkeiten: z. B. Messung niedriger Frequenzen über die Kehrwertbildung der Periodendauer, Phasenmessung mit zwei Bausteinen, Datenfernerfassung mit Spannungs/Frequenz-Wandler, schnelle Ereigniszählung und vieles mehr.

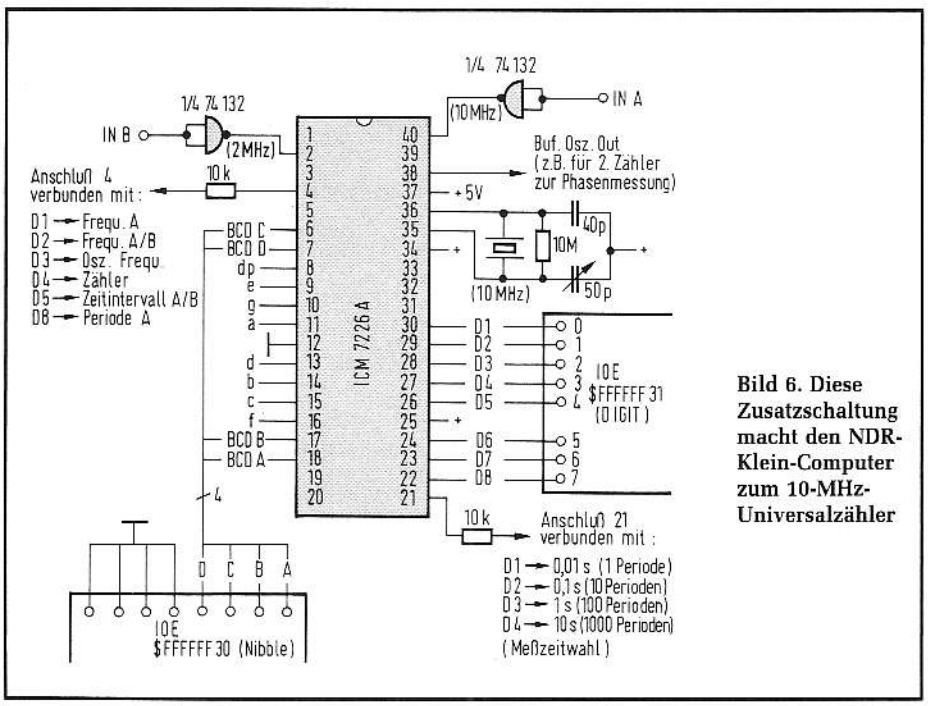

Bild 6. Diese Zusatzschaltung macht den NDR-Klein-Computer zum 10-MHz-Universalzähler

Mark und Release in Turbo-Pascal

Da im CP/M-Turbo-Pascal 2.00 für MARK und RELEASE ein eigener Heap-Pointer vorgesehen ist, dieser aber nicht von NEW und DISPOSE aktualisiert wird, muß man den für diese beiden Befehle benutzten Heap-Pointer in den Compilerroutinen von MARK und RELEASE ändern. Alle folgenden Änderungen beziehen sich auf die Datei TURBO.COM.
Die zu ändernde Routine von MARK findet sich, wie die *Tabelle* zeigt, unter der Adresse 1E63h. Dort wird der Heap-Pointer in eine Pointer-Variable gespeichert. Der umgekehrte Vorgang, also Speichern einer Pointer-Variablen im Heap-Pointer, läuft in der Routine von RELEASE bei 1E6Eh ab.
Die Systemvariable HEAPPTR liegt auf der Adresse 00CDh, und wird in der Variablen-Tabelle des Compilers bei der Adresse 726Eh definiert. Die besagte Änderung läßt sich am einfachsten mit DDT bzw. DDTZ und dem CP/M-Kommando SAVE ausführen, das Vorgehen ist ebenfalls in der Tabelle gezeigt. Es werden einfach alle (vorher fett gedruckten) CDs gegen EDs ausgetauscht. Nach dieser Modifikation funktionieren MARK, RELEASE, NEW, DISPOSE, GETMEM und FREEMEM im selben Programm einwandfrei. Auch das Testprogramm (*Bild*) läuft bis in alle Ewigkeit, ohne den Heap-Pointer zu erhöhen. *Th. Krull*

Tabelle: Lage und Änderung der betreffenden Speicherzellen

```
1E63  ED 5B CD 00  LD HL, (00CD)
1E6E  ED 53 CD 00  LD (00CD), HL
726E  CD 00 00 D2  "R "T "P "P "A "E "H

DDT TURBO.COM
S 1E65 ED
S 1E70 ED
S 726E ED
SAVE 119 TURBO.COM
```

```
var
   HeapPtr : integer absolute $00ED;

   p,q : ^integer;

begin
repeat
   mark(p);
   write(HeapPtr:8);
   new(q);
   write(HeapPtr:8);
   release(p);
   write(HeapPtr:8);
   new(q);
   write(HeapPtr:8);
   dispose(q);
   writeln(HeapPtr:8);
until false;
end.
```

Fehlerfrei und somit ohne Ende läuft dieses Testprogramm nur nach Modifikation des Compilers